Autonomous Tools and Techniques for Reducing Operational and Maintenance Costs in Space planes

Mark L. James
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
(818) 354-8488
Mark.James@jpl.nasa.gov

Wafa S. Aldiwan
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
(818) 354-4322
Wafa.S.Aldiwan@jpl.nasa.gov

Abstract—Our objective is to provide a collection of automated tools and techniques for reducing operational and maintenance costs in space planes. To demonstrate our capability, we used the AFE of JPL which will be flown on the X-33 as the testbed to examine our techniques. This paper discusses the tools that were used to develop that software. Another paper is in preparation to discuss the artificial intelligence being flown on the AFE.

One of the technology demonstrations on the AFE was to show how artificial intelligence could be used for reducing operational and maintenance costs. For this we used two JPL developed tools: Tspec and SHINE. Tspec was used for the automated testing and verification of portions of the AFE's software. SHINE was used as a high-speed inference engine for monitoring, analysis and diagnosis of portions of the AFE hardware. Faults are detected and diagnosed in flight with their resolutions logged by the X-33's.

This paper provides an overview of SHINE. A description of Tspec can be found from the references.

TABLE OF CONTENTS

2. BACKGROUND

Researchers in Artificial Intelligence (AI) of knowledge-based systems applications have faced a severe lack of sophisticated software tools running on flight hardware to assist them in developing AI approaches aimed at reducing operational and maintenance costs in space planes.

Much of the available commercial and "home-grown" software suffers from one or more severe limitations. The effect has been either to hamper the artificial intelligence programming techniques that could be used effectively in an application or to cause AI programmers to redevelop their own software tools to be used on flight hardware [2]. These limitations have historically included such problems as documentation, poor access to facilities at different levels, lack of modularity, poor run-time efficiency, inadequate debugging facilities and access to source code and lack of tools to support the most advanced reasoning techniques.

3. OBJECTIVE

Our objective is to provide a collection of automated tools and techniques for reducing operational and maintenance costs in space planes. To demonstrate our capability, we used the Avionics Flight Experiment (AFE) of Jet Propulsion Laboratory (JPL) which will be flown on the X-33 as the testbed to examine our techniques.

The X-33 is a joint program between NASA and Lockheed. They have partnered to produce the X-33 test vehicle to demonstrate advanced technologies that will dramatically increase reliability and lower the cost of putting a pound of payload into space from $10,000 to $1,000. The X-33 program will demonstrate in flight the new technologies needed for a Reusable Launch Vehicle (RLV).

We used two JPL-developed tools as examples of how AI can be used for reducing operational and maintenance costs and we used the AFE as our testbed for testing out our ideas.

The Test Specification Language (Tspec)[1] was used for the automated testing and verification of portions of the AFE's software and hardware. Spacecraft Health Inference Engine (SHINE)[7] was used for monitoring, analysis and diagnosis of portions of the AFE hardware. Both these tools were used on the AFE during system Integration and Test (I&T) and while in flight. Faults are detected and diagnosed in flight with their resolutions logged by the Vehicle Health Manager (VHM) of the X-33 through the 1773 system bus.

4. AVIONICS FLIGHT EXPERIMENT AS A TESTBED

The AFE of JPL is being flown as part of the X-33 as a testbed to demonstrate new hardware and software for reducing the cost of space planes. Because the AFE was designed and built by JPL, this provided us many opportunities to influence the hardware design and capabilities of the end system. This means that the AFE provided us with a very rich environment for testing out our ideas.

The hardware of the AFE is composed of a PPC 603ev processor clocked at 200 MHz, 512KB of level 2 cache, 196 MB of RAM, 220 MB of non-volatile memory, six redundant 1553 flight buses, one 1773 optical bus, an avionics package including of a Geo Positioning System (GPS), accelerometers, gyro data, A to D converters and timers.

The software was composed of the following major modules:

1. **Boot System Software**
   The bootstrap component provides three capabilities: nominal booting, off-nominal booting and internal visibility.

2. **Boot System Software**
   The bootstrap component provides three capabilities: nominal booting, off-nominal booting, and internal visibility.

3. **1553 Device Driver**
   Interfaces to the six redundant 1553 buses to provide an unidirectional data flow of all X-33 1553 bus traffic to the AFE.

4. **1553 Bus Data Parser**
   A programmable facility which can be dynamically configured to extract specific measurements from 1553 bus traffic, convert them to the specified units and send them as individual measurements to the Data Routing Task for distribution.

5. **1553 Resource Manager**
   Interfaces to the 1553 device driver to provide a unidirectional data flow of 1553 bus data from the 1553 hardware to the resource manager.

6. **Data Routing Task**
   Responsible for receiving data from the 1553 and the sensor assembly and sending it to the various AFE tasks via the IPC facility. It also provides an interface to supply the Avionics Health Manager with selected real-time 1553 data using a unidirectional pipe from the Data Router to the Avionics Health Manager.

7. **1773 Device Driver**
   Interfaces with the ASCENT AS-1773 Bus Controller chip. The AFE acts as the bus controller on the 1773 bus, and the X-33 VHM subsystem is the only Remote Terminal on that bus. The DD1773 encapsulates the hardware details of the ASCENT chip, and provides a simple Read/Write interface to the VTM task.

8. **Sensor Assembly Driver**
   The hardware driver interface to the AFE's Sensor Assembly Module. This is the software interface to the GPS and accelerometers.

9. **Sensor Assembly Resource Manager**
   Interfaces to the Sensor Assemble Driver to acquire real-time sensor assembly data, convert the data into engineering units and send it to the Data Routing Task.

10. **Non-Volatile Memory Manager**
    A silicon hard disk that is used to contain the AFE software and store the configuration information for each flight.

11. **VHM Transaction Manager**
    Provides a bi-directional interface to the 1773 optional bus of the X-33. This bus is used to record the results from AFE on the X-33 for later processing by ground operations.

12. **Enhanced State Estimator**
    The AFE has an internal Sensor Assembly Board containing a micro-gyro (planned), micro-accelerometers, and GPS devices. The ESE task receives input data from the Sensor Assembly Manager task, which handles the sampling of the AFE sensors and coverts the raw data to engineering units (for the gyros and accelerometers).

13. **Watchdog Timer**
    Provides the basic capability of having the AFE Software automatically reboot itself if the watchdog task stops working or the CPU halts.

A diagram of the relationship between the avionics health manager relative to the AFE software can be seen in figure 1.

5. Discussion of Tspec

Tspec[1] is a behavior specification language, a compiler, and a library of support software that together enable users to specify expected behaviors, compile those into auditor modules that are included within a C/C++ system under test, and get notifications when behavioral expectations are violated. These embedded behavior auditors analyze the [lengthy] observed behavior in real time to verify if the application logic is respecting all constraints.

Given appropriate specifications, the Tspec auditor can detect a variety of misbehaviors. These include:

14. Detecting invalid values (e.g., Transmitter A voltage should never exceed 40 volts);
15. Liveness violations (e.g., Missing heartbeat from state estimator);
16. Resource violations (e.g., Task A on CPU B is consuming too many resources);
17. Missing values (e.g., Missing update from gyro);
18. Illegal state transitions (e.g., Subsystem A goes from "Off" to "Off" without an intervening "On");
19. Duration violations (e.g., Battery heater that stays on longer than is expected);
20. Invariant violations (e.g., Taking a sensor reading when the cover is closed);
21. Out-of-sequence events (e.g., turning on an exciter before turning on its power amplifier).

Testing with Tspec is centered on declarative specifications of acceptable behavior, in the form of invariants, state machines, episodes, and constraints. Behavior encompasses values of observed measurements, update frequencies, event sequences, interval durations, repetition rates, and flight rules. Compared to methods that check for specific forms of misbehaviors, this approach is easier because users specify a relatively small number of correct/expected behaviors rather than a huge variety of misbehaviors, and more robust because it flags behaviors that deviate from the specifications.

Specifications of expected/acceptable behavior are expressed in this higher level user-oriented language and compiled into a lower level language which is combined with the Tspec library to create the test auditor module in a system under test.

The Tspec language attempts to satisfy two communities of users. First, for users who will be encoding behavior specifications (like flight rules) that come from documents and conversations, the language should be simple and intuitive enough that it can be learned in about an hour and it should allow for incremental accumulation of and refinement of behavior specs. Second, for projects whose flight software executes formal plans, the language must be expressive enough to encode such plans so that plan execution can be verified automatically.

Tspec is similar in expressive power to linear temporal logic, but its forms (invariants, state machines, episodes, and resource constraints) are believed to be more readily understandable and usable by spacecraft system engineers than are the temporal logic *future* operators (Next, Always, Sometime, Until).

6. Discussion of SHINE

SHINE is a reusable inference engine for the monitoring, analysis and diagnosis of real-time and non-real-time systems. It is intended for those areas where inference speed, portability and reuse are of critical importance. When the knowledge base is cross-compiled, its resulting size is extremely small and it can easily fit on targets with limited memory.

Knowledge acquisition and implementation from experts is an inefficient and painful process for most automation implementation projects. This phase is often so difficult, that the success of the automation project as a whole is jeopardized, and then often the resulting system is too slow or large to fit on the target system. SHINE was designed to address such problems by providing an efficient, with respect to speed and size, development and delivery environment.

SHINE is not intended to be an all-encompassing inference system. For those applications requiring advanced inference strategies and representational capabilities, then another inference engine would be more appropriate.

SHINE was designed to solve the usual kinds of monitoring and diagnostic problems found in flight projects, factory automation or the general area of intelligent sensor monitoring. In those cases then it provides a very effective and efficient solution for the representation and execution of such problems.

When a problem can be defined in terms of its attribute stimulus representation (discussed later), then SHINE provides a cost-effective approach to large-scale distributed software systems because of its data flow representation of rules. This reduces the complexity of the conflict-resolution match cycle by the transforming the knowledge base into a data flow diagram. The data flow diagram is then translated to the destination target programming language for efficient representation and execution.

The inference cycle never needs to pause for system-level activities such as garbage collection because the final representation preallocates all necessary storage for the inference process. This provides the knowledge base with a much more predictable execution profile that is often necessary in real-time applications.

SHINE has contributed to reduced operations cost, improved reliability and safety in eight NASA deep space missions that include Voyager, Galileo, Magellan, Cassini and Extreme Ultraviolet Explorer (EUVE). SHINE has been delivered to the NASA's X-33 as a component of JPL's Avionics Flight Experiment (AFE) and will be flown in 1999.

7. THEORY OF OPERATION OF SHINE

SHINE runs on multiple platforms and it has been ported to PCs, MACs, SUNs, VAXes and the AFE's flight computer. It is fully reusable and portable requiring less than three hours to port from one machine to another that supports Common LISP. In addition, it contains cross compilers for translating a knowledge base to C and C++ without any reliance on the LISP environment. SHINE is originally based on the STAR*TOOL [8][9] and rewritten to be more efficient and sensitive to flight processor requirements and limitations.

Rules are translated into stimulus/response objects that are then woven into a data flow model. The execution speed is improved from a sophisticated mathematical transformation based on graph-theoretic data flow analysis. The data flow representation is then transformed into threaded procedures for rapid execution.

Figure 2 shows the steps involved in the SHINE compiling process. SHINE takes a knowledge base composed of attribute and variable descriptions, forward and backward chaining rules and function definitions described in a common representation language and generates code.

One target that is always generated is the development environment and an optional target of one of the language target generators, e.g., C, C++, etc.

SHINE includes a high-speed development environment that allows for the easy definition, editing, testing and delivery of knowledge-based systems. When speed is of critical importance, a cross compiler is seamlessly integrated in the development environment to translate the knowledge base to C or C++. Cross compilers for ADA and JAVA are planned for 1999.

The development environment allows you to incrementally define your attribute and variable descriptions, forward and backward chaining rules and function definitions and test your system. This environment is written in LISP and executes very efficiently.

If execution speed and size are of critical importance, then the entire knowledge base can be cross-compiled into one of the optional targets. The result code is very small without any reliance or emulation of the original LISP representation. The resulting code is often small enough and efficient enough to fit on 8-bit microprocessors.

Representing rules as stimulus/response objects not only enhances the forward chaining inference process. It doesn't enable pattern-directed goal retrieval, i.e., backward chaining. Backward chaining inference is implemented by a transformational system that rewrites the backward chaining rules into their corresponding forward chaining counterparts.

SHINE replaces traditional inefficient pattern-based rules with collections of stimulus/response attributes containing constraints of arbitrary complexity.

SHINE contains a collection of compilers which translates the rules through a series of phases ultimately resulting in the target code, e.g., machine code, C/C++, ADA (delivery in 1999) or JAVA (delivery in 1999).

SHINE replaces relations with objects called attributes. They are like variables in that they have values and they can be grouped to form relations. They are unlike variables in that rules that access their values get scheduled for running whenever the attribute is assigned a value.

Attributes can have complex constraints associated with them including all the usual relational, logical, functional and mathematical operators.

Forward and backward chaining rules are used to represent knowledge processes where the attributes are used to represent the information. The activation of a rule is based upon the attributes in a rule being assigned a value and the constraints upon those attributes holding true at that instant in time. Any rules containing those constraint-satisfied values will be scheduled for execution. This process is repeated until there are no more rules that can be executed and the Inference process ends.

A rule may have computational and inference side-effects:

- Computational side effects are the typical computational algorithmic effects, e.g., assignments, calling functions that change something.
- Inference side-effects are those which modify something in the knowledge base which causes a rule to be scheduled for evaluation, i.e., the assignment of a value to an attribute.

8. PREVIOUS SHINE DELIVERIES

Some of its successful areas of application include:

1. Spacecraft Health Automatic Reasoning Pilot (SHARP) for the diagnosis of telecommunication anomalies during the Neptune Voyager (VGR) Encounter. Several hours before the encounter it detected a failing transponder. This was detected long before it was possible by human operators which prevented possible down time during the most critical phase of the mission.

2. Galileo (GLL) mission for diagnosing problems in the Power and Pyro Subsystem (PPS).

3. Magellan (MGN) mission for diagnosis of telecommunication anomalies in the TELECOM subsystem.

4. Engineering Analysis Subsystem Environment (EASE) which is an operations environment to operate a large number of spacecraft simultaneously, maintain high reliability levels and increase productivity through shared resources and automation.

5. Extreme UltraViolet Explorer (EUVE) mission for labor 3 to 1 shift reductions through the use of artificial intelligence.

6. Being flown on NASA's X-33 as part of JPL's Avionics Flight Experiment (AFE) for monitoring and diagnosis of 1553 redundant bus failures and flight phase and mode identification.

7. Fault Induced Document Officer (FIDO) for the EUVE mission: an automated system that assists in expert knowledge acquisition, access and publishing capabilities for safely managing complex systems under staffing reductions and "lights out" operations.

8. Being evaluated by Welch-Allyan for detecting and classifying colon cancer.

9. Johnson and Johnson is evaluating SHINE for the control of a robotic system that performs endoscopic surgery.

10. Being used by mission operations to diagnose anomalies in the Deep Space Network (DSN) Antenna Array systems by providing a solution for optimal DSN decision-making by integrating analytical and artificial intelligence methods.

9. CONCLUSIONS

Just as artificial intelligence can play an important role in the monitoring and diagnosis of space planes, the tools that are used to develop these systems also play an important role. This is especially true when issues of reliability, real-time performance, limited code and execution size, ease of use and maintainability are all factored in.

The AI techniques and tools that were developed for the AFE are well suited for the monitoring and diagnosis of space planes and ground systems. Both SHINE and Tspec run well in environments where system resources such as processor cycles and memory are at a premium. Both of these systems have been demonstrated in stand-alone advisory systems for human operators as well as components of embedded systems. Both of the tools generate C++ code which allows them to run efficiently in flight systems with real-time operating systems such as VxWorks.

The benefits afforded by the application of these tools and techniques are significant. The architecture and autonomous fault diagnosis techniques pioneered in the SHARP[16][17] system have demonstrated important benefits for operator productivity and spacecraft safety and have the potential to reduce workforce requirements for future space operations.

These techniques are have their limitations. The diagnostic techniques developed for the SHARP system are most appropriate for highly complex spacecraft or ground systems where faults are not immediately diagnosable from surface behavior. Our tests have shown that some of the most common spacecraft anomaly situations are easily diagnosed by human experts and only simple, one-step inference is required. If this knowledge can be coded directly into a knowledge base, a simple heuristic associative diagnostic process may be preferable. It should be emphasized that building and testing knowledge bases is time-consuming and is the major bottleneck for application development[2].

No matter how good your tools are, knowledge acquisition remains a fundamental bottleneck for development of applications of these systems and for knowledge-base systems at large.

10. FIGURES

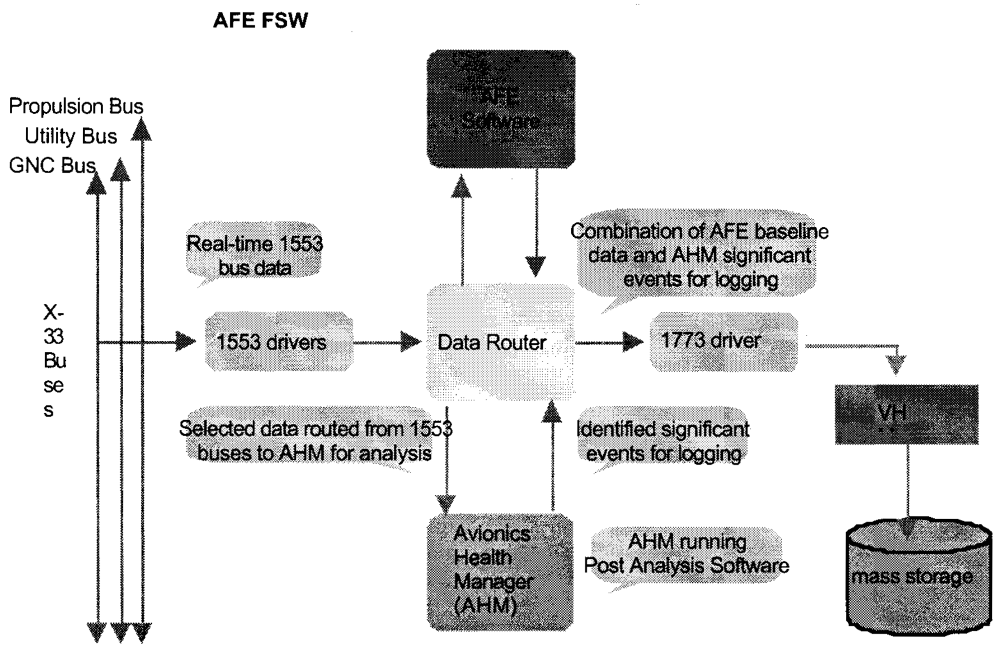
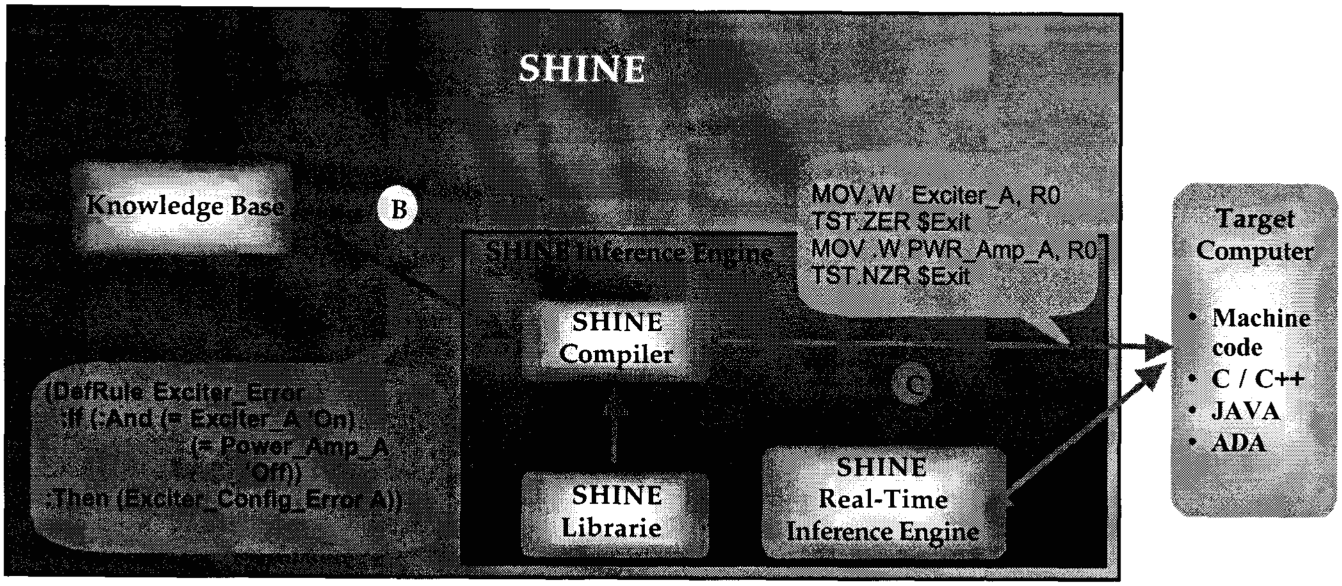


Figure 1: Block Diagram of Avionics Flight Experiment Software Data Flow



Knowledge bases are specified in a high-level programming language composed of common programming language constructs, e.g., If/Then/Else, For/While/Until, function calls.

B SHINE rules are sent to the SHINE compiler which links them with the SHINE libraries and then compiles them directly into native target code.

The machine code is linked into the target program space. The execution of the rules is controlled by the SHINE Real-Time Inference Engine.

Figure 2: Block Diagram of SHINE Internal Structure

11. ACKNOWLEDGEMENTS

The work described in this article was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

12. REFERENCES

[1] Lowry, Michael and Dvorak, Daniel. Analytic Verification of Flight Software, IEEE Autonomous Space Vehicles, 45-49, September/October 1998.

[2] Atkinson, David. Artificial Intelligence for Monitoring and Diagnosis of Robotic Spacecraft, Chalmers University of Technology, Sweden, Technical report 237, 1992.

[3] Abbott, Kathy H. Strategies and Representations for Onboard Aircraft Fault Diagnosis. SIGART Newsletter. Association for Computing Machinery. No. 92 April 1985a.

[4] Abbott, Kathy H. Exploration of Expert Systems Concepts for Onboard Diagnosis of Faults in a Turbofan Aircraft Engine. Proceedings of the American Control Conference. Boston, MA. 19-21 June 1985b.

[5] Abbott, Kathy H. Using Dynamic Behavior of Physical Systems for Real-time Fault Diagnosis: An AI Approach. IEEE Transactions on Systems, Man, and Cybernetics: Special Issue on Diagnostic Strategies. (Draft copy) 1986

[6] Davis, R. et al. Diagnosis Based on Description of Structure and Function. Proceedings of the National Conference on Artificial Intelligence. American Association for Artificial Intelligence. Pittsburgh, PA. 18-20 August 1982

[7] James, Mark L. *SHINE 5.7.4 Reference Manual* (JPL Internal document) August 16, 1998.

[8] James, Mark L. and Atkinson, David J. *Software for Development of Expert Systems,* NASA Tech Brief Vol. 14, No. 6, Item #8 from JPL Invention Report NPO-17536/7049 June 1990

[9] James, Mark L. and Atkinson, David J. *STAR*TOOL* from JPL Invention Report NPO-17536/7049 June 1989

[10] James, Mark, and Atkinson, David, "STAR*TOOL — An Environment and Language for Expert System Implementation", *Jet Propulsion Laboratory Report NTR C-17536*, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, August 19, 1988.

13. Biography

Wafa Aldiwan is a senior member of the technical staff in the Autonomy and Control section of the Jet Propulsion Laboratory in Pasadena, California .She has a Bachelor's degree in mathematics and a Master's degree in computer science from Texas A&M University at College Station, Texas. In 1985 she began work at Bell Laboratories. Initially she contributed in the design and development of a data network controller for the Datakit Virtual Circuit Switch, and received the prestigious Arno Penzias Award for her contributions. Later, she focused on system engineering of RND UNIX, a mainframe version of the popular UNIX System V operating system. In 1992 she transferred to AT&T's Operations Technology Center to work on software design and development of WMS, a large AT&T-internal client-server work management system. In 1996 she joined Jet Propulsion Laboratory in Pasadena, California and played a significant role (for which she received a NOVA award) in defining the process for hardware modeling of the Deep Space 1 spacecraft that was launched October of 1998. Currently, Wafa leads the software team for an avionics flight experiment scheduled to fly on X-33, a scale model prototype of the new reusable launch vehicle. In her spare time Wafa enjoys skiing, hiking and camping with her family.

Mark L. James is a Senior Member of the Technical Staff in the Ultracomputing Technologies Research Group in the Information and Computing Technologies Research Section within NASA's Jet Propulsion Laboratory, Pasadena, CA. At JPL Mark is Principal Investigator and Task Manager on a world class program in real-time inferencing and knowledge-based systems. He manages a number of JPL and NASA software projects. His primary research focus is on high-speed inference systems and their application to planetary and deep spacecraft systems, and, medical applications. His expertise includes core artificial intelligence technology, high-speed real-time inferencing systems, flight system software architectures and software design and implementation for those systems; image processing and scene analysis; pattern recognition, planning and simulation; real-time control systems, programming languages and compilers; symbolic mathematics, operating systems, hypermedia systems, AI programming environments, robotics, visual programming and virtual reality systems. Additional expertise in medicine, biology and analog and digital electronics design. Mark has received a number of NASA awards which include: NASA Software of the Year Award Nominee for 1997 and 1998, JPL Team Excellence Award for 1990, JPL JET Productivity Award for 1990, Major NASA Monetary Award for 1990, NASA Certificate of Recognition for 1990, NASA Achievement Award for 1990, NASA Exceptional Service Medal for 1989, NASA Exceptional Service Plaque for 1989 and NASA Manned Flight Awareness Nominee for 1988 and 1989.